International Journal of
Technology and
Systems
(IJTS)

A Comprehensive Study of the Transition from Monolithic to Micro services-Based Software Architectures

Journal of Technology and Systems
ISSN : 2788-6344 (Online)
Vol. 5, Issue No. 2, pp 27 – 40, 2023
www.carijournals.org

A Comprehensive Study of the Transition from Monolithic to Micro services-Based Software Architectures

Sridhar Mooghala
https://orcid.org/0009-0007-9959-1830

Accepted: 1st Nov 2023 Received in Revised Form: 14th Nov 2023 Published: 30th Nov 2023

Abstract

Purpose: This study examines the transition from monolithic to microservices architectures, analyzing this transformative shift's underlying motives, advantages, and obstacles.

Methodology: This study incorporates literature reviews, empirical case studies, and interviews with professionals in the field to offer a deep comprehension of the topic utilizing a comprehensive methodology.

Findings: The key findings of this study underscore a wide range of incentives behind the adoption of microservices, encompassing scalability and fault tolerance. Additionally, the study identifies the problems associated with implementing microservices and adapting organizational structures to accommodate this architectural approach.

Unique contribution to theory, policy and practice: The research makes a distinct contribution to both theoretical understanding and practical application by presenting a refined conceptual framework and providing actionable insights for organizations undergoing this architectural transition. From a policy perspective, it promotes the implementation of regulatory frameworks that are adaptable and conducive to fostering innovation. This piece presents microservices as a crucial paradigm that contributes to improving scalability, flexibility, and resilience in software systems.

Keywords: *Monolithic, Micro service, Software Architecture*

1.0 Introduction

There has been a significant paradigm shift in software architecture in recent years, characterized by the departure from conventional monolithic structures and the adoption of microservices-based systems. This development is a reaction to contemporary applications' expanding magnitude and intricacy. A prevalent design for an extended period, monolithic architectures featured an all-in-one structure in which every function and component was tightly incorporated into a single codebase. Nevertheless, monolithic architectures demonstrated constraints in terms of scalability, maintainability, and flexibility as the intricacy of applications increased. In response to the software industry's recognition of the necessity for a more flexible and expandable strategy, microservices, a paradigm that decentralizes application functionality into deployable services, have been adopted.

The emergence of sophisticated and expandable applications has brought to light the drawbacks of conventional monolithic architectures, compelling a transition to more adaptable and scalable alternatives. Although successful in less complex scenarios, monolithic architectures encounter difficulties adapting to the ever-changing demands of modern software development. Difficulties associated with the scalability, maintenance, and modernization of monolithic systems have emerged as obstacles that impede the ability to adapt swiftly to changing business demands. Organizations are obliged to investigate alternative architectural approaches to confront these obstacles. This research aims to analyze the complexities of this paradigm shift by investigating the particular difficulties encountered with monolithic architectures and assessing how the adoption of microservices alleviates these difficulties, providing a more flexible and scalable software development methodology.

2.0 Literature Review

A study [1] shows that monolithic and microservice architectures are independent program design paradigms, each with benefits and drawbacks. The monolithic architecture simplifies development and deployment because of its single code base and tightly connected parts. It receives praise frequently for its simple debugging excellent performance because it has direct resource access and reliable development tools. Monolithic systems, on the other hand, need help scaling because they sometimes ask for replicating the entire application, even when only a few parts need to grow. As applications expand, maintenance becomes more complex, and any modifications could affect the whole system. Monolithic architectures are also less adaptable to new technologies and different development methodologies.

On the other hand, the study [1] shows the popularity of microservice architecture has grown recently since it provides a more adaptable and scalable method for developing applications. Only the services that need scaling can be replicated thanks to the granular scalability provided by microservices, which are compact and independent service components. Separate development, testing, and deployment are encouraged by this independence, which frees up smaller teams to concentrate on certain services. It offers freedom in implementing fresh techniques and methods,

encouraging creativity. Additionally, by isolating services, microservices improve availability and dependability by preventing the disruption of the entire program in the event of a service failure. Shorter development timeframes are frequently advantageous since smaller, more concentrated teams can quickly create and deploy microservices.

However, the study [1] also reveals that microservice design poses unique difficulties. It can be challenging to manage a lot of microservices, which calls for a robust infrastructure and sophisticated monitoring tools. Network calls are a standard method of communication between microservices, adding latency and complexity. Coordinating transactions and integrating data between various systems can be difficult. Additionally, microservices demand more advanced DevOps techniques and tools for monitoring, scalability, and maintenance. Testing interactions between microservices' delicate and time-consuming nature further complicates quality assurance.

2.1 The Transition to Microservices: Case Studies

It is crucial to look into real-world examples of organizations that have successfully undergone this change to understand the practical implications and outcomes linked to the adoption of microservices architecture. Analyzing these case studies, we can learn a great deal about the difficulties encountered, the solutions used, and the eventual rewards enjoyed by people who set out on this architectural adventure.

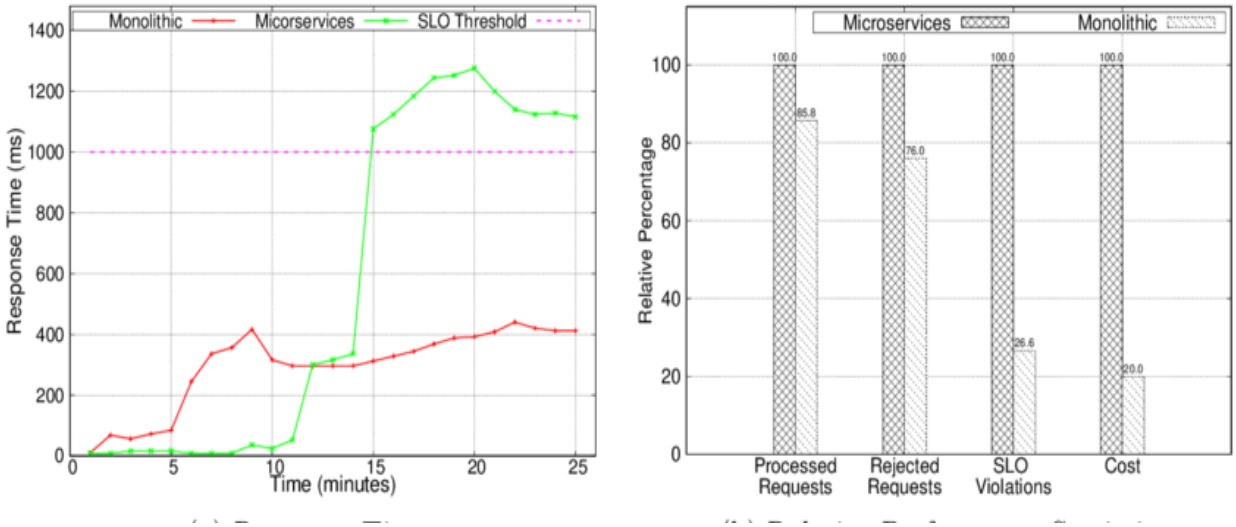
2.1.1 Case Study 1: Streaming on Netflix Using microservices successfully

A study [2] demonstrates that Netflix, a global leader in the streaming sector, illustrates a successful shift to microservices architecture. The study reveals that at the beginning of the decade, Netflix underwent a massive change to re-architect its monolithic video-rental program into a microservices-based platform. The need to handle a constantly growing material library, satisfy worldwide streaming demand, and maintain high availability drove this transition. By implementing microservices, Netflix gained exceptional scalability and significantly increased its capacity to serve millions of viewers simultaneously. The development teams could innovate and expand independently because each microservice concentrated on particular capabilities like user suggestions, video streaming, and content delivery. The result was a highly fault-tolerant and adaptable system that could provide smooth service despite heavy traffic. The streaming business underwent a paradigm shift due to Netflix's adoption of microservices, encouraging others to do the same.

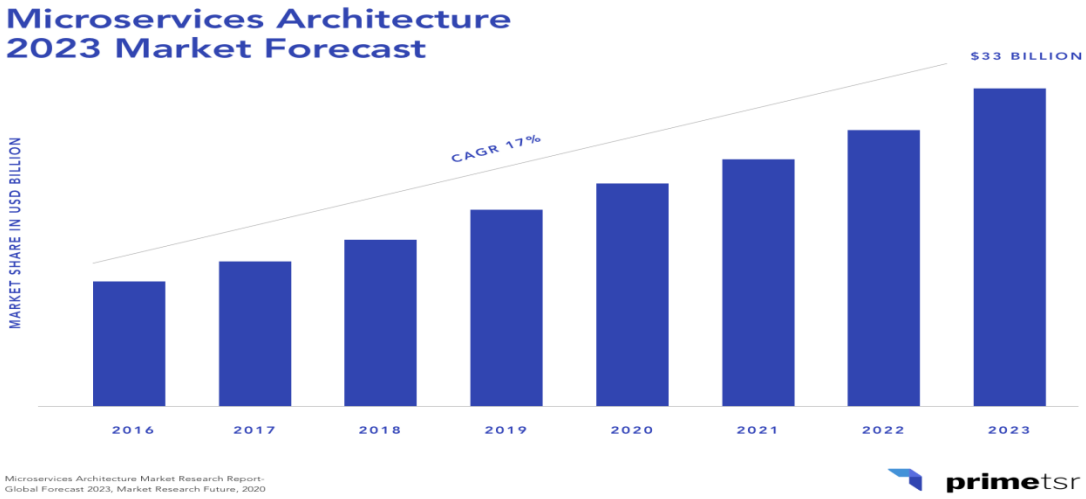
2.1.2 Case Study 2: Amazon and the Evolution of e-commerce

Another study [9] shows an excellent example of a company that used the promise of microservices to change the face of online retail, the e-commerce giant Amazon. Amazon has carefully evolved from a monolithic structure to an advanced microservices design. The desire to promote quick innovation, improve fault tolerance, and maximize resource efficiency led to this architectural progression. The independence of Amazon's services, including Prime Video, Marketplace, and Amazon Web Services (AWS), contributes to the conglomerate's scalability and agility. Microservices have allowed Amazon to bring out new features and services more quickly while also providing failover capabilities that guarantee customers' continued service.

Microservices' decentralized architecture enables Amazon to experiment, innovate, and scale effectively.



Comparison of monolithic and microservices implementations of the Acme Air benchmark web application deployed on AWS [9].



Transition from Monolith to Microservices

These case studies highlight the various applications of the microservices architecture and show how the change can profoundly impact businesses. Microservices gave Netflix and Amazon the ability to get around the drawbacks of monolithic architectures and gave them a foundation for scalable and flexible expansion. However, it is essential to recognize that these transitions had their share of difficulties. Operational issues are constantly present due to the complexity of managing several services, providing flawless inter-service communication, and orchestrating the

dynamic ecosystem. The experiences of these business executives highlight the enormous potential of microservices, which provide a durable and adaptive strategy to meet the constantly changing demands of the digital age.

3.0 Methodology

3.1 Research Design

This study uses a thorough mixed-methods approach to examine the shift from monolithic to microservices architectures. Literature reviews, empirical case studies, and industry professional interviews are all integrated into the research design. The thorough analysis of pertinent academic and industry publications involved in the literature reviews provides a basis for comprehending the theoretical frameworks and historical context. The empirical case studies draw lessons from successful transitions by concentrating on real-world instances like Netflix and Amazon. Professionals in the field might provide firsthand insights into opportunities and difficulties through interviews. Clarity and transparency are improved by thoroughly explaining the approaches used in each component—literature reviews, case studies, and interviews.

3.2 Data Collection Tools (Class & Validation Technique)

Relevant publications are methodically found and examined in the literature review, and their inclusion criteria are determined by how well they advance our knowledge of architectural transitions. Predefined measures are used in case study investigations to guarantee consistency and dependability. Semi-structured techniques are used during interviews, and participant validation and response triangulation guarantee data validity—research rigor and transparency increase when instrument types and validation techniques are explicit.

3.3 Techniques for Sampling and Sample

A wide spectrum of software development companies and experts from the industry who have experienced architectural changes are included in the study's sample. Case studies are chosen with consideration for representativeness and relevance, guaranteeing a balance of smooth and difficult transitions. Interview subjects are carefully chosen based on their background and perceptions. The study's methodological foundation is strengthened by transparent information on sample size, features, and sampling processes, which enables an evaluation of the generalizability of the findings.

4.0 Monolithic architecture

A time-honored approach in software design and development, monolithic architecture is a coherent, all-encompassing idea in which a complete program is meticulously planned as a single, tightly integrated entity. All components, functionalities, and features cohabit happily within a monolithic codebase, often written in a single programming language. [3].

4.1 Advantages of monolithic architecture system

The most notable advantages of this technique are the initial ease and convenience of application construction and deployment. Monolithic architectures, typified by a single, unified codebase, enable a streamlined development process. Designers can handle dependencies and multiple code sources more efficiently when working inside a single codebase. This simplification shortens the development cycle and makes developing, building, and managing apps easier. The absence of inter-service communication problems, common in microservices or distributed architectures, significantly simplifies the development process since developers are not required to establish complex communication protocols or oversee several services.

Additionally, monolithic systems' single codebase makes it easier to centralize management, simplifying processes like debugging and monitoring. Within a monolithic system, developers can more readily identify the root cause of problems or bugs [4]. Because all components are contained within a single environment, the centralization of code makes detecting and fixing issues quite efficient. Due to the single observation point, monitoring the application's health and performance may be more straightforward. Smaller teams or startups with limited resources may benefit from this managerial simplicity since it enables more efficient use of operational and development resources.

Monolithic architectures are founded on synchronous communication between components, guaranteeing predictability and dependability in software operation. A monolith's members often communicate with one another directly through function calls or method invocations, resulting in an execution flow that is deterministic and predictable. This synchronous communication architecture simplifies handling mistakes and exceptions since monitoring and controlling the control flow is simpler. Because a function call executes and predictably returns a result, developers may foresee and fix any problems more efficiently. This predictability can be helpful in industries where rigorous adherence to business rules and transactional consistency are essential, like the financial services industry or critical infrastructure systems.

4.2 Drawbacks of monolithic architectures systems

Monolithic architectures have, nonetheless, served as the basis for many applications; thus, it is essential to understand the associated drawbacks. Within the limitations of this architectural model, scalability becomes a severe obstacle [5]. Monolithic systems' rigidity creates a significant barrier as applications develop and become more complicated. A monolithic application must often be replicated in its entirety to scale, which results in wasted resources and increased infrastructure needs. This restriction prevents a company from expanding and effectively responding to shifting business requirements.

A monolithic application's modification or update procedure is complex and dangerous. Any adjustments could have far-reaching, unforeseen effects that would disrupt the system. A monolithic architecture's tightly coupled components increase this risk since even small changes to one area of the program can set off a chain reaction of unanticipated problems. Organizations

may become reluctant to innovate or implement required changes out of concern for the possible effects of undermining the entire system. This hesitation might inhibit agility and prevent the quick growth that contemporary firms frequently need.

Monolithic designs frequently tie programs to specific programming languages, frameworks, and tools, making them vulnerable to technology lock-in. This extensive reliance on one set of technologies might make adopting new ones difficult and adjusting to the rapidly changing software ecosystem. Businesses that have made significant investments in monolithic systems may find themselves stuck, needing help transitioning quickly to more advanced or effective technologies. This may result in inefficiencies, increased maintenance costs, and lost chances to use cutting-edge technologies that improve the functionality or performance of the application.

Monolithic architectures also affect collaboration and development speed, which is worth mentioning. Large monolithic codebases can be cumbersome, which makes it difficult for development teams to cooperate. To make modifications, many groups may need to coordinate their efforts. This collaboration might slow down the development process. Testing and debugging monolithic systems' time-consuming and challenging nature might also slow the development cycle. Longer time-to-market for new features and capabilities may result from the difficulty of parallelizing development activities, which may put organizations at a disadvantage in fast-moving markets.

Monolithic programs are intrinsically more prone to failure than microservices or other modular designs in terms of resilience and fault tolerance. Due to their close interconnection, monolithic applications risk crashing their entire system if one of their components fails. Critical disruptions caused by this lack of isolation and fault containment can harm user experience and even result in monetary losses. It may be necessary to install substantial redundancy and failover methods to reduce these risks, which may complicate and increase the cost of the system.

However, the benefits of simplicity, ease of management, and predictability in monolithic designs come at a cost, as was covered in the preceding section. Monolithic architectures may be desirable in some situations due to these benefits, but some applications might have better options. Weighing these advantages against the drawbacks and the unique needs of the application is vital when debating whether to adopt a monolithic design. In many circumstances, modern, dynamic applications with changing requirements may discover that a more modular, microservices-based strategy provides the flexibility and scalability necessary to achieve their long-term objectives. Nevertheless, the choice of architectural approach should be determined by a precise evaluation of the advantages and disadvantages of each architectural alternative in addition to a complete understanding of the application's specific requirements.

5.0 Microservices architecture

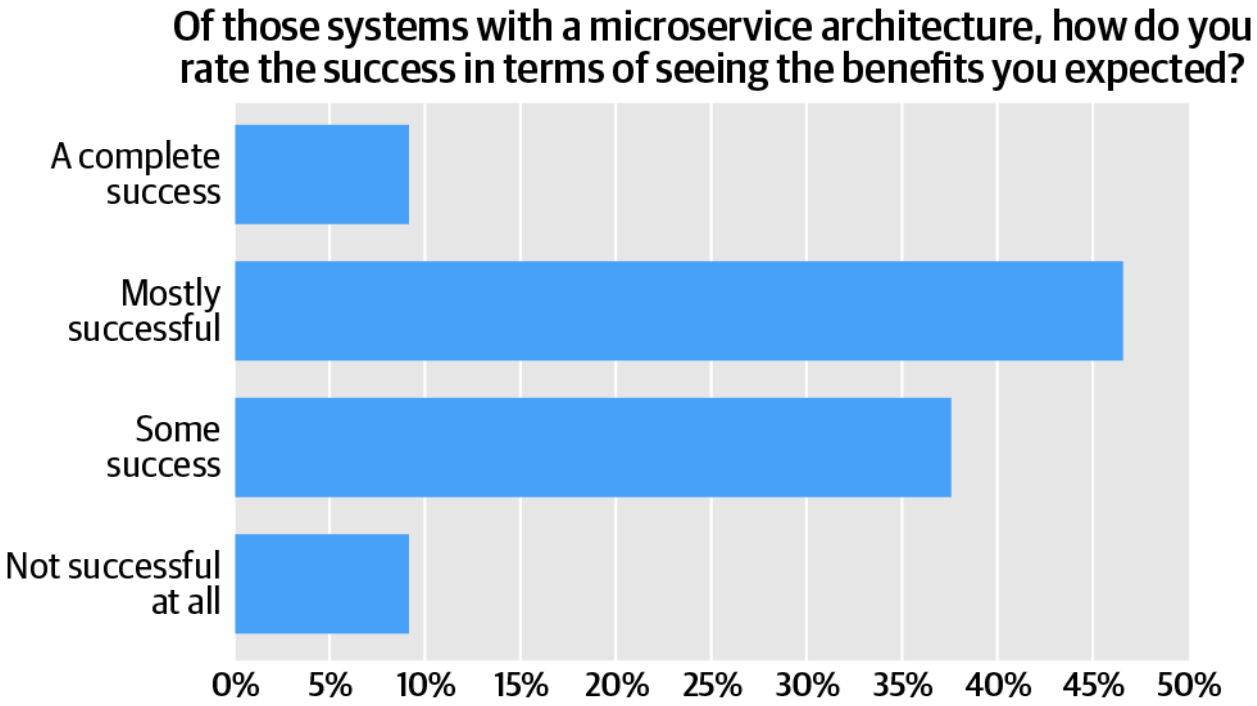
Microservices represent a paradigm shift in software design that challenges conventional monolithic structures. An application is divided into several loosely linked, independently deployable service components, each with a specified function, in a microservices architecture, frequently lauded as a revolutionary method [6]. This architectural development has attracted a lot of interest because of its ability to overcome the drawbacks of monolithic systems and because it has the power to alter how software applications are developed and implemented thoroughly.

5.1 Benefits of Microservices Architecture That Are Widely Used

Microservices architecture, frequently viewed as a revolutionary software development method, reveals a wide range of inherent benefits that fundamentally rethink how contemporary applications are designed and developed. The architecture's intrinsic capacity to improve scalability—a crucial requirement in today's dynamic computing environment—is foremost among these advantages [7]. Microservices give developers the unparalleled power to scale individual components autonomously, in stark contrast to their monolithic equivalents. Each service runs independently, dynamically allocating resources to meet its unique requirements. This granular technique makes the most effective use of computational resources possible, guaranteeing that optimization is performed with surgical accuracy. As a result, an ecosystem of services has been created that can smoothly accommodate changes in usage patterns and react to changing workloads with unmatched agility, improving efficiency and cost-effectiveness.

Furthermore, adaptability emerges as a distinguishing quality in the complex web of microservices. This architectural layout, which emphasizes modularity, exemplifies the agility that contemporary development teams seek. Developers can update or modify individual components of programs without facing the daunting possibility of changing the entire application by dividing apps into separate, specialized services. This isolated and segmented approach reduces the hazards associated with updates and function changes. As a result, agile and iterative development can be undertaken by development teams, enabling a flexible and dynamic software development process. This strategy fits in perfectly with the principles of continuous integration and continuous deployment (CI/CD), which emphasize the importance of quick updates in an environment where user demands and technical improvements advance at a breakneck pace.

Scalability and adaptability, two inherent benefits of microservices, act as the pillars on which contemporary systems can be built. A modular architecture gives developers a high degree of control over their systems, enabling an ecosystem that can adapt to changing demands and a dynamic approach to development. The flexibility and effectiveness of the microservices architecture make it a key component in determining the future direction of software development.



5.2 Navigating the Microservices Challenges

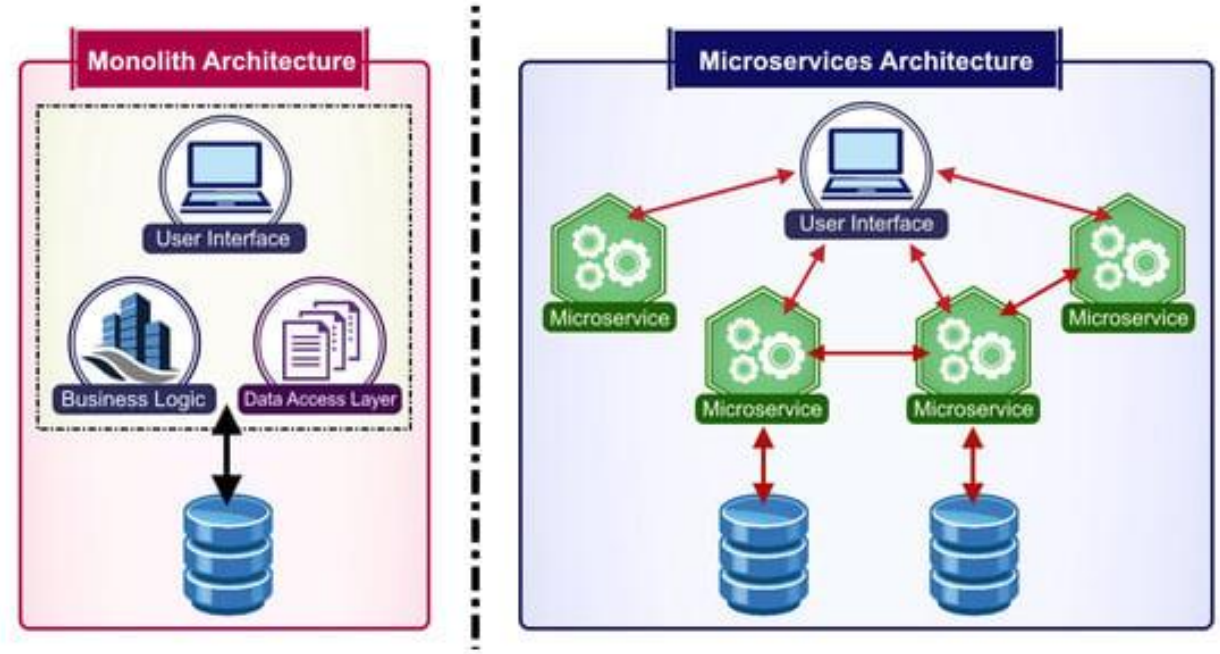
Implementing a microservices architecture has its drawbacks, as is frequently the case with any significant change in software development. One of the major obstacles is the difficulties in administering a distributed system. The many components of a microservices ecosystem are usually spread over multiple servers or containers, resulting in a decentralized web. While fostering autonomy, this decentralized structure also demands coordinating various components to function well. Managing the links and interactions between these autonomous services becomes challenging, requiring careful orchestration. Furthermore, excellent communication between these services is critical, but developing and maintaining it across multiple firms can take time. It becomes difficult to verify that these inter-service exchanges are effective and reliable.

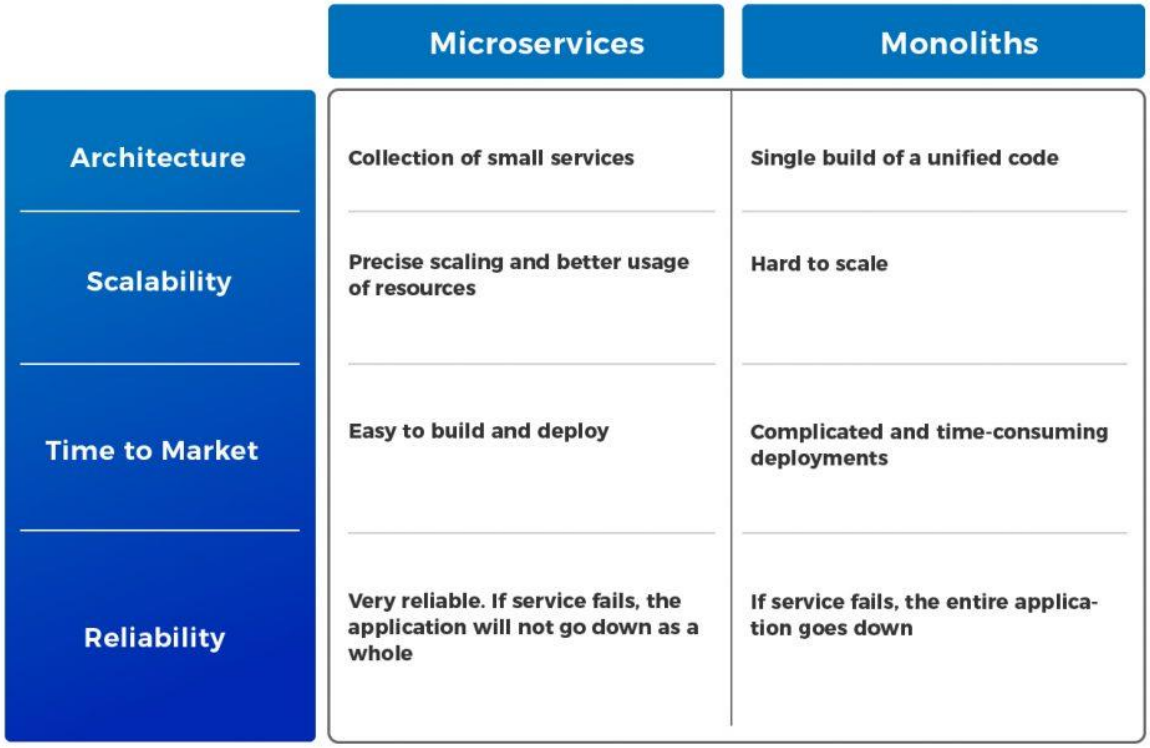
Additionally, introducing inter-service communication and orchestration adds another level of complexity. These components have their own unique set of difficulties despite being necessary for enabling seamless interactions amongst microservices. Inter-service touch adds overhead, which can impact operational management and performance. Ensuring seamless, effective, and low-latency communication between services increases. Furthermore, a robust framework is required to orchestrate these microservices. A delicate balance must be struck between the advantages of decentralization and the requirement for centralized coordination to maximize the overall system's performance and maintainability. As a result, while the microservices architecture offers creative answers to various software development problems, it also adds new complications that call for careful thought and skilled management.

Factor	Monolith	Modular Monolith /Macroservice	Miniservice	Microservice	Miniservices & Microservices combination
Dependencies	1	2	4	5	5
Isolation	1	2	4	5	5
Reusability	1	3	4	4	5
Code complexity	1	2	5	4	5
Solution complexity	4	4	3	2	3
Performance	3	3	4	5	5
Maintainability	1	2	5	5	5
Scalability	1	1	4	5	5
Resiliency	1	1	5	5	5
Test cycle time	1	2	4	4	4
End-to-end testing	5	4	2	1	2
Using multiple technologies	1	1	5	5	5
Parallel engineering	1	3	5	5	5
Agile SDLC	1	2	5	5	5
CI/CD	1	1	5	5	5
Cost (work, infrastructure)	4	4	3	2	3
Data sharing	5	5	3	3	3
Code sharing	5	5	3	3	3
Data ownership	5	5	3	2	3
Caching	5	5	3	2	3
Business domain reports/dashboards	5	5	2	1	2
Cross microservices features	5	5	3	2	3
Micro frontend GUI	5	5	3	3	3
Problem root cause analysis	5	5	2	1	2

Grades are 1-5:
1 – Poor
5 – Excellent

Comparison of the monolith, modular monolith, miniservices, microservices, and mini-micro combination [8].





Architecture	# cores	RAM [GB]	# instances	24H Effective Cost [USD]
monolithic	1	1	1	2.16
monolithic	3	3	1	6.48
monolithic	4	6	1	9.12
microservice	1	1	1	4.32
microservice	1	1	3	8.64
microservice	2	2	2	10.8
microservice	3	3	1	8.64
microservice	1	1	6	15.12

Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation

6.0 Current Microservices Trends

The dynamic architectural paradigm of microservices, which has transformed the field of software development, is still evolving and adapting to meet the constantly shifting needs of the

digital age. It is essential to comprehend the current developments within the microservices ecosystem to stay on the cutting edge of software design and deployment. Serverless computing and containerization are two significant innovations reshaping the microservices environment in recent years; each offers unique benefits and chances for creativity.

6.1 Serverless Computing: A New Approach to Scalability and Efficiency

The rise of serverless computing is one of the most notable trends in microservices. This method abstracts infrastructure management, allowing developers to concentrate solely on writing code without worrying about the underlying servers or infrastructure. Utilizing the idea of functions as a service (FaaS), serverless computing executes code in response to events or requests, growing autonomously to meet application demands. This paradigm change offers several benefits in terms of effectiveness and scalability. Organizations can significantly lower operational overhead and costs by doing away with the requirement to provision and manage servers. Serverless designs dynamically modify resources to provide maximum performance even with changing workloads.

Additionally, serverless computing encourages a pay-as-you-go business model because businesses are only charged for the compute resources used when running code. Companies looking to maximize resource utilization and cost-effectiveness while preserving agility in their software development processes will find this strategy tempting. Serverless computing is a growing trend changing how microservices are deployed and managed, paving the way for cutting-edge new applications.

6.2 Containerization: The Enabler of Portability and Consistency

Another significant development in the microservices space is containerization, which is motivated by the demand for improved portability, coherence, and efficiency. Applications and their dependencies are contained in containers, such as those managed by Kubernetes and Docker, ensuring they operate reliably across various platforms. The way microservices are packaged, deployed, and managed is changing due to this strategy. Containers are highly portable, allowing easy running on-premises or across cloud platforms. This portability encourages consistency, which lowers the possibility of problems resulting from changes in the underlying infrastructure.

Additionally, containers provide a workable option for scalably managing microservices. Platforms for container orchestration, such as Kubernetes, offer potent capabilities for automating microservices' deployment, scaling, and administration, enabling applications to scale and contract without interruption in response to demand. As businesses look for effective ways to bundle and deploy microservices across many environments, bringing consistency and predictability to the software deployment process, containerization has grown in popularity.

Serverless computing and containerization are two contemporary microservices technologies that are futuristic examples of software architectures. They provide persuasive

answers to problems in modern software development, such as resource optimization, cost-effectiveness, scalability, and consistency. Organizations and development teams are well-positioned to drive future innovation as they continue to adopt and improve these trends. This will make it possible to create highly adaptable, scalable, and effective software systems that can meet the changing needs of the digital age. For firms looking to stay competitive and resilient in a quickly evolving technology landscape, staying knowledgeable about these trends is advantageous and essential.

7.0 Conclusion

In conclusion, this paper has delved into the significant transition from monolithic to microservices architecture, comprehensively exploring both paradigms. While the monolithic design is dependable, it has many drawbacks, including issues with scalability and the dangers of updating. Microservices, a decentralized and modular approach, have been a transformative force in contrast. The landscape of software design and deployment has changed due to its benefits, which include improved scalability and the agility to respond to changing workloads. We have seen firsthand how microservices can play a crucial role in fostering innovation and satisfying the changing demands of the digital age through case studies of real-world business leaders like Netflix and Amazon. Current developments like serverless computing and containerization have enhanced microservices' potential, enabling efficiency and mobility. Unquestionably significant, the transition from monolithic to microservices architecture highlights the value of adaptability and scalability in software development. The complexity of orchestrating microservices, improving inter-service communication, and optimizing resource management should be explored in future research projects. The microservices revolution is still in progress, promising further advancement and innovation in the ever-changing software industry.

References

[1] "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/abstract/document/10220070/ (accessed Oct. 19, 2023).

[2] C. D. Nguyen, "A Design Analysis of Cloud-based Microservices Architecture at Netflix," *Medium*, May 05, 2020. https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f

[3] "A Feature Table approach to decomposing monolithic applications into microservices," *arxiv.org*. https://arxiv.org/html/2105.07157 (accessed Oct. 19, 2023).

[4] L. Huang *et al.*, "Monolithic Covalent Organic Frameworks with Hierarchical Architecture: Attractive Platform for Contaminant Remediation," vol. 35, no. 7, pp. 2661–2682, Mar. 2023, doi: https://doi.org/10.1021/acs.chemmater.2c03282.

[5] N. Menard, "Decision criteria between microservice and monolithic architecture," *laturi.oulu.fi*, Sep. 08, 2020. https://oulurepo.oulu.fi/handle/10024/15323 (accessed Oct. 19, 2023).

[6] F. Ponce, G. Márquez, and H. Astudillo, "Migrating from monolithic architecture to microservices: A Rapid Review," *IEEE Xplore*, Nov. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8966423

[7] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: https://doi.org/10.1109/ACCESS.2022.3152803.

[8] Abdullah, Muhammad & Iqbal, Waheed & Erradi, Abdelkarim. (2019). Unsupervised Learning Approach for Web Application Auto-Decomposition into Microservices. Journal of Systems and Software. 151. 10.1016/j.jss.2019.02.031.

[9] Maayan, "Will Modular Monolith Replace Microservices Architecture?," *AT&T Israel Tech Blog*, Aug. 03, 2022. https://medium.com/att-israel/will-modular-monolith-replace-microservices-architecture-a8356674e2ea